RC25174 (W1106-060) June 17, 2011
Computer Science

IBM Research Report

Polygraph: System for Dynamic Reduction of False Alerts in Large-Scale IT Service Delivery Environments

Sangkyum Kim
UIUC

Winnie Cheng, Shang Guo, Laura Luan, Daniela Rosu
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 704
Yorktown Heights, NY 10598

Abhijit Bose
Google

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

Polygraph: System for Dynamic Reduction of False Alerts in Large-Scale IT Service Delivery Environments

Sangkyum Kim
UIUC
kim71@illinois.edu

Winnie Cheng, Shang Guo,
Laura Luan, Daniela Rosu
IBM Research
{*wcheng,sguo,luan,drosu*}*@us.ibm.com*

Abhijit Bose
Google
abose@google.com

Abstract

Today's large-scale IT service delivery systems encompass multiple data centers, geographical locations, diverse hardware and software platforms. Services are no longer confined to racks within a single data center — they may often be deployed and served from multiple locations. Further, with the increasing adoption of virtualization and cloud computing, the management of large-scale IT infrastructure is increasingly the focus for data center optimization and innovation. Of the various service management tasks such as incidents, problems, changes, and patches, handling of incidents is often a major portion of the work performed by the system administrators managing the system components. In this paper, we focus our attention to monitoring alerts which are triggered by agents monitoring the health of the system components based on pre-set thresholds. A fraction of these alerts get converted into service tickets that must be investigated and resolved within a specified time duration. Our data, collected from a very large IT service environment, as well as previous studies indicate that a significant portion of these incidents can be false, often as high as half the total volume of alerts, resulting in wasted work investigating them.

In this paper, we describe Polygraph, a system for reducing false alerts and incidents. Polygraph works by mining historical incidents and alerts, and by correlating them with other historical data such as system health time-series data, server similarities, operational context of servers and other sources. The resulting output is a set of monitoring policies with projected accuracies and rates of false alert reduction if deployed in the environment. Polygraph is unique in that it uses an active learning approach with these outputs. It presents policies with projected low scores to system administrators for verification instead of automatically deploying them. Polygraph uses a four-step process: In the first step, it attempts to detect alerts that can safely be removed (i.e. false alerts); second, it generates a set of candidate policies to achieve this purpose by estimating new thresholds; next, it calculates, via simulation, a projected savings in terms of false alerts that would be removed from the environment while ensuring that true incidents are not missed; and finally, upon verification by system administrators, these new policies are dispatched to monitoring servers that further push them to individual components. We evaluate Polygraph with a real-life trace of around 60K incidents collected over 30 days from a portion of a large IT service delivery infrastructure. We divide the older traces for learning purpose, and use the more recent traces for testing the effectiveness of the new policies generated by Polygraph. Our results indicate significant reduction of false alerts while keeping the number of missing true events (i.e. false negatives) to a minimum. We also discuss several ways Polygraph can be extended to increase its false alert detection rates and overall effectiveness.

1 Introduction

Efficient management of IT operations and facilities is a major competitive advantage for service providers, given the massive scale and costs involved with today's IT Service Delivery Infrastructures. In these environments, massive physical infrastructures (networking, power, cooling, security) exist to deploy and manage server farms, as well as run applications for different clients. System and application incidents and failures occur almost 24x7 and an incident management framework must be in place to respond to them in a timely manner and in accordance with customer Service Level Agreements (SLAs) and delivery Service Level Objectives (SLOs). Proactive prevention and in-time response to failures with minimal operational costs is a major target for service providers. Substantial resources are required for monitoring systems and managing incidents. Depending on the size and complexity, managing the IT

infrastructure's operations can cost companies billions of dollars.

This paper addresses the problem of effective response to incidents with an automated approach that dynamically customizes the failure and incident detection mechanisms. It does so by integrating inputs of diverse types (incident reports, monitoring alerts and policies, system vitals, system configuration, service level objectives) from diverse operational domains (e.g., customers, clusters) and by applying machine-learning algorithms. This integrated approach allows our system to effectively reduce the number of alerts that can be safely ignored, also called *false alerts*, thereby reducing the need for human action.

1.1 Background

Incident management systems such as IBM Tivoli 7 [8] and HP ServiceCenter 7 [7] are examples of conventional approaches to handling the logging of monitoring alerts and incidents, dispatching them to appropriate system operators, and tracking their resolution. Furthermore, the timeliness in resolving these issues is critical, as IT service providers and clients have Service Level Agreements that specifies the maximum time-to-resolve for issues with different severity levels. In addition to SLAs, Service Level Objectives are attached to specific system components. SLOs specify specific targets, e.g. 99.99% availability for a group of mail servers. Failure to meet SLAs and SLOs results in financial penalties and damages the relationship with the clients.

For effective system management, components are configured with one or more performance monitoring agents. The agents monitor a range of key performance indicators (KPIs) to assess the health of the component. They rely on pre-defined policies to trigger the creation of alerts. These policies typically specify thresholds for one or more KPI values, possibly combined with an execution context. In most monitoring frameworks, the higher level nodes may perform alert reduction and incident ticket generation based on policies that aggregate in time and space (i.e., across multiple systems). Eventually, System Administrators (SAs) receive the filtered alerts and auto-generated incident tickets requesting action.

The monitoring policies for the managed system components have a significant impact on the scalability of the monitoring infrastructure: the costs due to SA resources spent on incident management, and the costs due to SLA and SLO penalties. Policies may trigger alerts at times when there is no actual critical situation. For instance, this may happen when policies for KPIs such as CPU, memory or file system utilization, are set at too low thresholds or too short intervals for observation of out-of-policy behavior. These false-positive signals represent non-value-added work for the SAs and in fact, from our discussions with many of them, for some datacenters, this represents a major portion of the total alert volume.

In order to balance scalability, costs and SLA/SLO risks, monitoring policies must be defined based on a detailed understanding of component behavior — the default policies and thresholds that come configured with the component are not always optimal. This is in itself a hard goal to achieve. First, as pointed out in [3], appropriate thresholds on operational parameters are seldom known in advance; system behavior can change often, as servers and applications/middleware can be dynamically configured to serve different purpose/clients and periodically re-imaged with different operating systems and system configurations. Second, the operational costs of modeling system behavior and composing new policies can be very high due to time and skill requirements. Third, policies are limited by the capability of the monitoring agent to instrument and assess KPIs.

As a result of these difficulties, the typical approach in datacenters is to setup the vast majority of servers with default policies that are far from optimal, yet serve well the need to minimize SLA/SLO failures but at a higher cost. Monitoring agents are setup with best-practice packages of policies that are generally customized for the system management services (such as security, recovery) that are deployed throughout the infrastructure, and the middleware installed on the system (e.g., databases, application servers). As an initial assessment, we studied the effectiveness of this approach for a large IT Service Delivery organization and found significantly high percentage of false alerts [1]. Our study also revealed that the lack of systematic approaches for tuning the policies, the high variability of workloads and the absence of continuous process improvement programs were key reasons for such high percentage of false alerts in these environments.

We propose *Polygraph*, a system that helps reduce false alerts by automating system behavior modeling, dynamic policy generation and dynamic deployment. Polygraph employs machine-learning techniques to model system behavior and assess the effectiveness of policies. The novelty of Polygraph draws from two principles:

- **Learning from the SAs:** Polygraph learns from the resolutions of historical incident tickets handled by SAs , and about the alert instances that could be safely ignored. By temporal correlation with system vitals (more generally, KPIs), configuration details, change management events, Polygraph creates a profile of the managed component that is used for automated refinement and deployment of monitor-

[1] The exact numbers were omitted due to client confidentiality

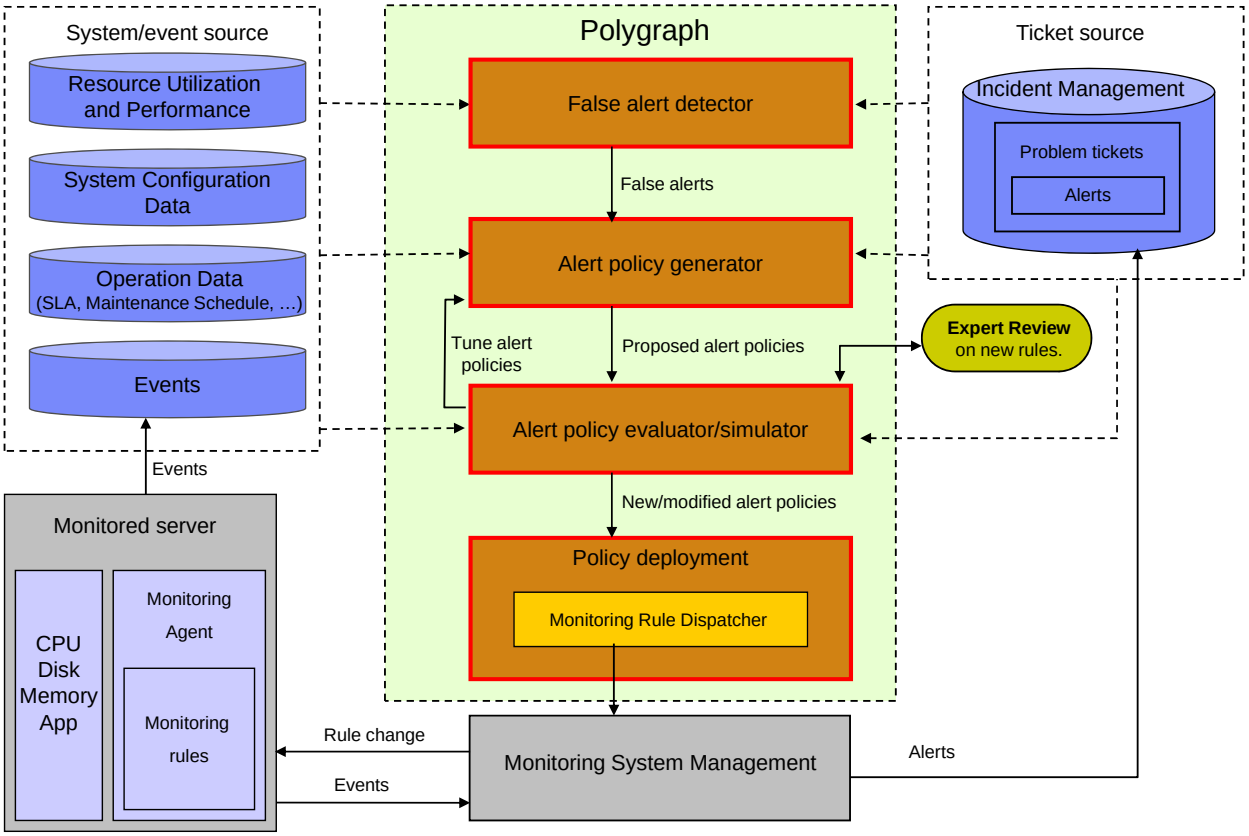

Figure 1: Polygraph System Architecture and Environment.

ing policies. This principle enables effective assessment of policies without elaborate, resource consuming direct system analysis.

- **Leverage Component Similarity in Large Scale Environment:** Polygraph exploits its assessment of similar components to expand the input size for its learning tasks for improved accuracy. Furthermore, policy changes are deployed to groups of similar components rather than individual components one-at-a-time, thus increasing the likelihood of reducing false alerts even for servers that have not experienced them in the past.

Figure 1 illustrates the integration of Polygraph within a typical service management infrastructure and its main components. In addition to its close interaction with the monitoring infrastructure, Polygraph leverages data from several system management tools, including configuration management databases, repositories of historical system vitals, incident management, and change management systems.

Polygraph comprises of the following main functionalities: 1) analysis of alert effectiveness and identification of false alerts, 2) generation and evaluation of new monitoring policies, and 3) deployment of new policies by interacting with monitoring system management. The Polygraph prototype described and evaluated in this paper is focused on the analysis of events and generation of new policy. The implementation uses the IBM Tivoli policy specification language.

The evaluation is based on a large set of monitoring events, incident tickets, and system vitals from a large IT service provider organization. The evaluation compares several approaches for generation of new monitoring policies.

Prior research in the area of incident management has focused primarily on anomaly detection [12, 4, 3] and patch management [13, 10]. Anomaly detectors can help automate the identification of problem areas (e.g., unexpected bursts in network traffic from an infected or ill-written application). The published studies suggest a very similar situation in that many of these incidents/alerts are often false positives. Most of the previous works have provided valuable insights into the management of expected and unexpected behaviors for specific domains (e.g., enterprise applications, databases, or networks). They aimed at differentiating abnormalities from baseline or historical behavior by employing

sophisticated algorithms to mine these patterns. Our focus on understanding system behavior overlaps with anomaly detection, but we are interested in detecting issues that occur from normal day-to-day operations in a data center, for example, the need for additional storage for a server. Patch management associates system configuration changes with problems, in an attempt to automate the problem resolution process. Such techniques are also effective in reducing the costs in managing data centers and are orthogonal to our work.

Overall, the main contributions of our work are:

1. Design system architecture that can support the continual refinement and assessment of policies based on effective exploitation of diverse types service management data,
2. Propose techniques for identification of false alerts by mining historical incident resolutions,
3. Propose techniques for generation of new monitoring policies that can significantly reduce the volume of false alerts and evaluate the effectiveness with a large sample of data.

The paper is organized as follows. In the next section, we discuss related work and contrast them with our approach. Section 3 describes Polygraph architecture and Section 4 provides implementation details. Evaluations of the Polygraph prototype are presented in Section 5 with a follow-on discussion in Section 6. Finally, we summarize key findings and discuss future work in Section 7.

2 Related Work

The relevant literature for enterprise alert management can be grouped into three major areas: infrastructure monitoring (including security), data mining and analysis of monitoring alerts and finally, techniques targeting false alert reduction.

There are several well-known commercial and community-supported platforms for monitoring datacenter and IT infrastructure components. For example, IBM Tivoli Monitoring [9], Nagios [15], HP OpenView [7] are used by thousands of enterprises to monitor critical IT infrastructure components, such as servers (CPUs, Fans, Disk Drives, Memory), OS, rack- and cluster-level network infrastructure, service protocols, applications and middleware. As described earlier, monitoring agents are configured with policies for generating alerts based on observed KPI levels on monitored components. Most of the monitoring products support alert-suppression based on a statically specified policy. Our work is different in its use of dynamically generated policies based on mining the alert streams and other service management data. With the increasing adoption of virtualization and cloud computing in datacenters, major vendors have recently added agents for monitoring virtual machines and cloud infrastructure. Many well-known cloud service providers have developed and deployed custom monitoring and alert-response systems as part of their unique infrastructures. Our work does not depend on a specific monitoring system or technology, and the ideas presented can be extended to any IT Service Delivery environment.

An apparatus for alert prioritization on high-value end points such as automobiles and appliances is described in [1]. An aggregator agent is employed to receive monitoring events from end point agents and processes them locally to determine the priorities based on the rules and the associated environment information known to the local aggregation agent. The aggregator agent communicates the prioritized events to a central system to reduce false alerts and improve efficiency. The work focused on the functionality of the aggregator agent and did not address dynamic policy settings from data mining of historic event data.

[3] is the closest to the present study and describes an algorithm for automated and adaptive threshold setting based on Service Level Objectives (SLOs) of applications. The basic idea is modeling of SLO violations for applications dependent on a set of components using logistic regression and subsequent retrospective adjustment of the alert thresholds for these components. However, the complexity of the approach increases significantly with multiple applications, components, SLOs, and their combinations. Also, for an infrastructure with large footprints and heterogeneous systems, it is still difficult to find repositories of application-component dependency graphs and SLOs specified at component levels, despite the progress made by many IT service providers in this area.

By far, the largest volume of published literature on enterprise alarm management is on network monitoring, and in particular, intrusion detection system (IDS) alerts. We refer to [2] for a thorough review of IDS-related work, and discuss a few studies relevant to our work. IDS alert data mining, such as episode mining [17] and clustering [12, 11], are used to analyze historical alarms, to find alert root causes to help eliminate future redundant alerts, and to identify false-positive alerts. In general, IDS alerts contain only a real or false indication of attack, which is a different model from the performance monitoring alerts considered in this paper, that include quantified resource usage levels such as percentages of disk or CPU utilization. As a result, false-alert detection methods used for IDS alerts are not relevant for our work. More relevant is the window-based state monitoring method in [16] that evaluates whether a state violation is continuous within a given time window, and

therefore gains immunity to short-term value bursts, e.g. threshold violations and performance outliers. This can reduce false alerts and the overall overhead associated with alert counter-measures.

In [5], the authors describe an approach based on frequent itemset detection [6] to identify false positives in IDS alerts. They argue that the attributes of most frequent itemsets represent 'normal' data (i.e. common configuration problems and not real attacks). Each IDS alert is treated as a transaction, and each alert attribute is treated as an item. The frequent item patterns are used to extract the features of false positives, and these features are to filter alerts. This approach can potentially be applied to resource monitoring alerts, as well, because on only a small fraction of the alerts represent a true system or resource utilization issue.

An IDS decision support system for construction of an alert classification model for on-line network behavior monitoring in IDS is proposed in [14]. The proposed system architecture comprises a three-phase alert analysis method - alert preprocessing phase for alerts sequencing and correlation, model constructing phase for construction of rule classes used to classify and filter false alerts, and, a rule refining phase for adaptive classification of an alert sequences across different time intervals of an alert sequence. This proposal is similar. The approach of building classification models and rules using historic data mining is also applicable to service delivery infrastructure addressed in our work. However, more diverse data sources should be integrated in order to construct meaningful and effective monitoring rules for server operation alerts.

We conclude our review with an interesting viewpoint on the difficulty in classifying rare events [4]. In the present context, an example of a rare event can be a combination of server and middleware alerts indicating a service being unavailable on rare occasions. Such events are difficult to classify with traditional learning classifiers, which are often biased for the most common events. The authors argue that if the events are rare and not too costly, the learning algorithms can do little to improve. If the events have a much higher cost, then a large number of false alarms must be tolerated. This is exactly one of the best-practices in IT Service Delivery for management of high-risk SLO.

3 Polygraph System Architecture

In large-scale IT Service Delivery, the system monitoring infrastructure [9, 15, 7] is vital for ensuring that all IT components perform at levels that minimize the risks of SLA and SLO failures, financial penalties and maximize client satisfaction. Monitoring agents are deployed to monitor KPI values (e.g., CPU utilization, availability, web application response time, number of DB connections) for each IT component and signal situations that are relevant for the overall performance and SLO risk. Such signals are identified based on monitoring policies that include conditions related to KPI values, and time and operational context (e.g., day of the week and time in the day, active processes, and log entries). These signals are captured as monitoring events.

The system-wide monitoring infrastructure propagates and aggregates monitoring events, and, eventually, provides input to SAs in the form of alerts or incident tickets. SAs analyze the alerts/tickets and take corrective action if necessary. However, often, no corrective action is needed because the alert was a false-positive in the identification of critical situations. Such alerts are called 'false alerts' or 'ignorable alerts'. False alerts occur when the monitoring policies fail to capture sufficient detail on the parameters governing a critical situation or when imprecise thresholds are set overly conservatively in fear of missing high-risk SLA/SLO failures.

For a competitive IT service delivery, false alerts must be minimized in order to reduce operational costs. However, our observations and study of related work [2] identify the difficulty of achieving these goals in complex execution environment. In our observation of a large-scale service delivery infrastructure, the volume of false alerts can be significant, varying with the maturity of the delivery operations.

In practice, the service providers define monitoring policies based on best-practices for the specific configuration and service management features of the monitored components. False-positives can be reduced with detailed analysis and fine-tuning of monitoring policies. However, even with state-of-the-art tooling, this approach requires substantial SA effort and skills, prohibiting its large-scale adoption. Furthermore, SAs have limited system visibility and cannot easily share lessons-learned from other SAs in geographically-dispersed locations.

Polygraph, the framework proposed in this paper, specifically addresses these limitations with the automation of monitoring policy evaluation and fine-tuning. Figure 1 illustrates the integration of Polygraph in the service delivery infrastructure and its main components. The goal of Polygraph is to identify false alerts and design new monitoring policies that lowers the occurrence of false alerts while preserving negligible SLA/SLO failure risks. Towards this end, Polygraph integrates novel methods and exploits content that originates from numerous service-management components, such as incident management, configuration management, change management, system vitals, monitoring events and alerts, and operation data (SLAs/schedules).

Polygraph comprises of four functional components

(see Figure 1):

1. **False Alert Detector,** performs the analysis of current alert specification effectiveness and false-alert detection;
2. **Monitoring Policy Generator,** performs the generation of monitoring policies based on observed false-alert patterns;
3. **Monitoring Policy Evaluator,** performs the evaluation and adjustment of newly generated monitoring policies; and
4. **Monitoring Policy Deployment,** performs the deployment of new monitoring policies by close interaction with the monitoring infrastructure.

In addition to these components, Polygraph includes component for providing and receiving input from SAs. Polygraph can provide reports about observed false-alarm patterns, and justification for its decisions of policy generation and deployment. Polygraph can receive guidance for handling of situations where data quality is limited and high-quality automated decisions are not possible.

The novelty of Polygraph methods draws from two principles:

- **Learning from the SAs.** Polygraph is learning from the resolutions of incident tickets handled by SAs in the past, about the alert instances considered to be ignorable (false alerts). By temporal correlation of these insights with system vitals, configuration details, change management events, Polygraph creates profiles for the monitored system components, and uses them for new policy generation, assessment and deployment. This principle enables effective assessment of policies without an explicit system analysis which can be highly elaborate and resource-consuming [3].
- **Leverage Component Similarity.** Polygraph exploits monitored-component similarity with respect to configuration and operational context, monitoring policy and false alert patterns components. This input is used in multiple ways for improving the quality and scalability of new-policy generation. Component similarity is used to aggregate across alert input in order to achieve large data sets and improved learning accuracy. Furthermore, policy changes are deployed to groups of similar component, thus reducing the amount of work required to improve performance of every server, and reducing the occurrence of false-alerts on servers likely to exhibit behavior already present to similar servers.

In the following, we discuss each Polygraph component.

3.1 False Alert Detector

Polygraph must be able to distinguish false alert from true alerts. It does this by learning from SA's assessment of incident resolutions. The False Alert Detector has two main functions. First, it must be able to identify the incident tickets that are generated automatically by monitoring systems. Second, it must be able to assess whether the identified alert corresponds to a real problem or is a false alert. These tasks can be complex since the details of incident records do not provide this information in structured fields and free-text mining is necessary to extract the information. Section 4 describes the algorithms we used to achieve such functionality when free-text mining is used. The detector performs the ticket classification off-line and the results are later used by the Alert Policy Generator to determine how policies should be modified.

3.2 Alert Policy Generator

The Alert Policy Generator is the core component in which off-line data mining techniques are applied to fine-tune policies. Some examples of alert policies include the triggering of alerts when certain system parameters rise above or fall below pre-defined thresholds. The choice of techniques to be used for this task depended on the constructs available in the policy language. Section 4 goes into the details of the techniques we explored for our IT service delivery environment based on IBM Tivoli. The result of the analysis is a set of new policies, such as threshold changes and extensions of conditions to capture time and operational context.

3.3 Alert Policy Evaluator

The Alert Policy Evaluator assesses the impact of newly generated sets of policies by simulating these policies against historical alerts and events data. The goal of the simulation is to assess the potential of false-alert reduction and the missed true alerts that might result due to algorithmic fine-tuning. The results are available for review by SAs, with details on their accuracy and SLA impacts. The system administrators can choose to accept or reject these recommendations. This fine-tuning can be an iterative process, generating new policies and evaluating them.

Polygraph-specific policies can be defined to enable the automated deployment of newly generated policies if the accuracy and SLA impacts are above thresholds defined by SAs. This allows Polygraph to reduce SAs' effort for less complex and critical investigations, which account for a majority of the incidences. The policy evaluation can be limited to a specific monitored component

or can encompass a large set of servers based on predefined similarity criteria. For instance, similarity can be defined by server type (Web server, DB server, etc.), the hardware configuration (CPU, memory, disk type), applications installed, server execution context (maintenance/security process), SLOs or other operations data, resource utilization patterns, or geographical location. This approach builds on current best-practices of deploying the same package of policies on similar servers. The benefit is an increase in the effectiveness of false alert reduction.

The accepted policy changes are placed in a policy deployment package that is passed to the Policy Deployment component.

3.4 Policy Deployment

The Policy Deployment module interacts with the underlying monitoring infrastructure to deploy the newly generated policy packages. This module uses the internal specification of new policies to generate the necessary details about rule deployment specific to the monitoring infrastructure. This module implements appropriate staging of requests for policy deployment to minimize disruptions on the monitoring infrastructure.

4 Component Details

In this section, we describe the Polygraph prototype, referring to the methods and algorithms used to implement the main components of the framework: false-alert detection, alert policy generation, and alert policy evaluation. The scope of the current prototype is limited to threshold-based alert policies. In this context, the automated generation of new policies is equivalent to automatically adapting the threshold values of each rule.

4.1 Detecting False Alerts

Polygraph is a learning-based framework to utilize historical data including server resource utilization and performance metrics, system configuration data, incident events, and alerts. Mining patterns or correlations among these data enables the detection of false alerts.

Among all incident events, in the present framework, we only consider threshold-based alert policies which cover the majority of issues. For this purpose, these alerts are separated from other human-entered tickets by searching for centain special signatures in the problem description, which are added by the policy engine when issuing the alerts. The signatures have well-defined structured and entity-value expressions. An alert contains several fields including incident occurred time, problem description, and solution description (which is available after the incident is resolved). The historical data of these machine-generated alerts are classified based on their solution fields. First, we classify whether a ticket is an alert (automatically generated) or is manually entered. Second, of the ones that are alerts, we categorized them into two types: True or False. False alerts are defined to be the ones that are (1) duplicates, (2) not reproducible, or (3) have no action taken in their resolutions. These classified alerts are passed to the next module for analysis.

An alert can be triggered by one or more incident events, and an event is generated by an alert policy. In general, an alert policy is defined by any user-specified predicate (indicator function) over the performance metrics. A monitoring rule may specify a counter threshold, and require an alert to be sent when such threshold is met.

There are three basic alert policy types: (1) IF A; (2) IF A AND B; and (3) IF A OR B. Here, A and B are predicate units consisting of one parameter and its corresponding threshold value. The first alert policy is the smallest form of a predicate unit, while the other two alert policies are either a conjunction or a disjunction of two predicates of the first type. The following examples illustrate three basic types of an alert policy.

1. IF ($System.Virtual\_Memory\_Percent\_Used > 90.0$)
2. IF ($NT_Physical\_Disk.Disk\_Time > 80$) AND ($NT\_Physical\_Disk.Disk\_Time <= 90$)
3. IF ($SMP\_CPU.CPU\_Status =$ 'off-line') OR ($SMP\_CPU.Avg\_CPU\_Busy\_15 > 95$)

As the third alert policy example shows, different parameters can be used in the same alert policy, and they are compared either with numeric values or with categorical values. Polygraph, in its current form, focuses on numeric threshold values wrapped in an inequality condition, but still considers the cases where multiple performance metrics are used in one alert policy. For the rest of this subsection, we describe two dynamic threshold adaptation schemes for the basic alert policy types. In practice, an alert policy is a complicated composite of these three basic types of predicates, and our schemes easily apply to these more complex cases because every predicate can be converted into a conjunctive normal form. These schemes target to reduce the number of events that generate false alerts, which would result in reducing the number of false alerts as well. Without loss of generality, let us assume that a higher parameter value means a higher workload of a server.

There is one important necessary condition that must be satisfied when tuning the threshold of an alert policy.

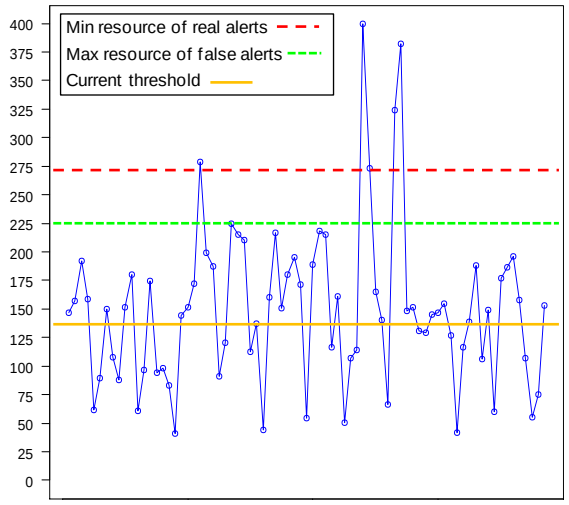

Figure 2: An example of a sequence of performance metric values and their corresponding alerts.

The new threshold must not generate any false negatives. That is, we do not want to miss any true alerts by tuning the threshold, which makes the problem hard to solve. We assume that the given history data (or training data) for learning purpose is large enough to ensure high accuracy of the Polygraph framework, and we will later show that this assumption is realistic by experimental results of various sizes of training data in the evaluation section.

Figure 2 illustrates an ideal case of tuning a threshold for a basic type (1) alert policy. If the parameter values of true events (events that generated true alerts) are all above the parameter values of false events (events that generated false alerts), then we can set up the new threshold to be the minimum parameter value of true events. This new threshold will divide events (and therefore alerts) into two categories, and perfectly detect all false events. But, in the real life, this case seldom happens. We need to allow more general cases into considerations, not to detect all false events but to reduce the number of false events as much as we can, which is described in the following proposition. We define a *true set* of an alert policy P to be the set of parameter values of true events which were triggered by P.

Proposition 1 *For a given alert policy P consisting of one parameter p and its corresponding threshold θ, let T be the true set of P, and let $t = min(T)$. If $t > \theta$, then we can set a new threshold of P to be t and it will generate no false negatives for the given dataset.*

Proposition 1 describes how to tune the threshold value for a basic type (1) alert policy. If the smallest performance metric value t that triggered true alerts is bigger than the original threshold value, then we can safely set it up to be a new threshold value θ'. As an example, given a policy governing CPU threshold, if all true alerts happen for thresholds greater than or equal to 95%, we can safely raise the original threshold of 90% to 95%. But even if we change the threshold based on this scheme, there will be no gain on doing this if the smallest performance metric value that triggered false alerts is bigger than or equal to t, since all events are still above the new threshold θ' and no new false alerts can be detected by θ'. That is, if we have a false alert happening at threshold 98%, this does not affect our new threshold setting. As mentioned above, note that we assume that we have enough history data (or training data) that guarantees all different performance patterns that caused the true alerts.

The next proposition describes a method to tune threshold values for basic type (2) and (3) alert policies.

Proposition 2 *For a given alert policy P consisting of either a conjunction or a disjunction of two predicate units A and B where A has one parameter p_1 and its corresponding threshold θ_1, and B has one parameter p_2 and its corresponding threshold θ_2. Let T_1 and T_2 be the true sets of A and B of P respectively. Let $t_1 = min(T_1)$ and $t_2 = min(T_2)$. If we set the new thresholds θ_1' and θ_2' to be t_1 and t_2, then they will generate no false negatives for the given dataset.*

Suppose that P is a disjunction of two predicate units A and B, and P is triggered by values $p_1 = a$ and $p_2 = b$. If the value $p_1 = a$ contributes to triggering A and thereby triggering P, then we add a to T_1, but if the value $p_2 = b$ does not contribute to triggering B (and therefore to P) we do not add b into T_2. In case P is a conjunction of two predicate units A and B, and P is triggered by values $p_1 = a$ and $p_2 = b$, then they should always be added to their corresponding true value sets, since for a conjunction of two predicates, both of them have to be satisfied at the same time to trigger the conjunction predicate.

For the rest of the paper, we consider an alert policy to be in the basic type (1) for the purpose of simplicity.

4.2 Mining Event and Server Characteristics

At the time of deployment, when the servers and their sensors were initially set up, most of the servers were clustered together based on their workload types and the alert policies were consistently determined for each cluster of servers. As the enviornment changes over time, there can be a significant change in each server's workload. Therefore, sometimes the servers need to be upgraded by increasing their memory and hard disk capacity, or sometimes one needs to change their configurations based on the applications they run. Their initial

clusters may have moved into a larger cluster or divided into smaller clusters. Even within the same cluster, the servers may now be performing different tasks that redefine the boundaries of these clusters. Similarly, the initial settings of alert policies on each server which were originally set-up in a group-wise manner, now need to be tuned separately for themselves based on their own configurations and workloads. For example, if we increase a server's hard disk capacity from 100GB to 10TB, then a threshold of 90% will generate alerts when it is using 9TB of disk space even though there are 1TB of free space left.

For these reasons, Polygraph tunes alert policy thresholds for each server and our initial results show that one can gain trmendously from this approach.

We do not exclude the possibility of re-clustering the servers based on current environments and tune their alert policy consistently within the same group, we leave this as future work since defining a similarity measure based on multiple attributes of the servers is out of the scope of this paper. A server's profile becomes a complex data type because its time-stamped performance metrics represent a (numeric) time series data, its configuration is textual data, and its time-stamped events and alerts are (textual) time series data.

4.3 Extrapolating Time-Dependent Behavior

To further improve our false alert detection framework, Polygraph takes into account time dimension of the data since events, alerts, and performance metrics are all time-stamped. In general, there are two ways to utilize time dimension of the data: (1) mining time-related patterns of false alerts; and (2) mining time-related patterns of true alerts.

The idea of mining time-related patterns of false alerts comes from the observation of periodic patterns of daily jobs, weekly jobs, or even monthly jobs that can significantly affect memory and CPU usage of a server to be captured by some of its alert policies. These routine jobs should be considered as false alerts. Finding periodicity is a key technique for this approach. For example, in Figure 3, a green solid circle shows a daily pattern and a red dashed circle shows a weekly pattern of a performance metric time-series data. A drawback of this approach is that even if we mine all time-related patterns for false alerts, we cannot completely consider them all as false alerts since there is a chance that a true event occurs together with them. For this reason, Polygraph follows the second approach of utilizing time-related patterns of true alerts.

Given an alert policy P, Polygraph mines a set of true ranges for each host, because Polygraph performs false

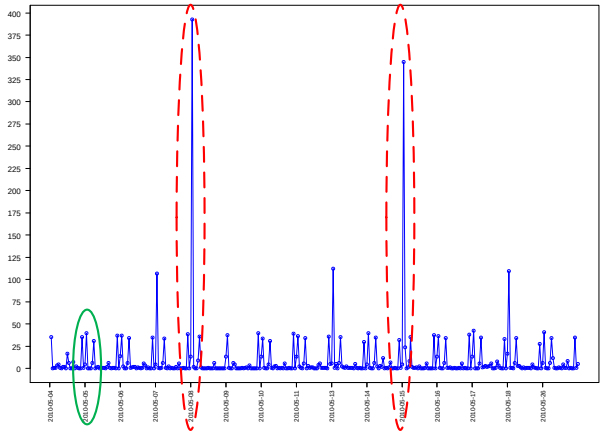

Figure 3: An example of a sequence of performance metric values that contains daily patterns and weekly patterns.

alert detection for each host as mentioned in the previous section. A *true range* is a range of performance metric values that might trigger P. It does not mean that every value in a true range always triggers P, but it means that all candidate values triggering P fall in this range. For an alert policy P, it might have one or more true ranges for each host.

Similar to most of the frequent pattern mining algorithms in data mining that needs to specify a minimum support to retrieve frequent interesting patterns, Polygraph requires a user-specified threshold to decide the width of a true range to mine a set of these true ranges for each alert policy.

Polygraph mines a set of true ranges in the following way. First, Polygraph scans the history data (or training data) to get a true set T for each host H. Given the true range threshold, Polygraph constructs a set of true ranges R from T. Once an event of the host H occurs outside the true ranges R, then consider it as a false event.

Proposition 3 *For a given alert policy P, let T be the true set of P, and R be its true range. Then, for the given dataset, all parameter values that trigger true alerts of P will be in R.*

For example, suppose host H has three true events at 3pm, 4pm, and 8pm. Given a true range threshold of 1 hour, two ranges (2pm-5pm) and (7pm-9pm) become H's true ranges. If the threshold becomes smaller, Polygraph will detect more false alerts but there might be higher risk of missing true alerts. In the evaluation section, we show the effectiveness of our approach with various settings of true range thresholds.

4.4 False Alert Prevention Algorithm

Based on the above false alert detection schemes, Polygraph helps to significantly reduce the number of false alerts and resulting manual investigation by system administrators. We describe the procedures of the false alert detection algorithm in Algorithm 1.

Algorithm 1: False Alert Prevention

Learning Phase
Input1: History data of performance metrics, events, and alerts
Input2: A true range threshold δ
Output: True sets and true ranges
begin
1. Scan DB
2. **for** each rule and host
3. Construct true sets
4. **for** each true set T
5. Construct T's true ranges

end

Prediction Phase
Input1: Event E occurred at host H
Input2: True sets and true ranges
Output: Filtered alerts
begin
6. **If** (E's parameter value is below min(T(H))
7. **then** E is a false event
8. **Elseif** (E's parameter value is outside R(H)
9. **then** E is a false event
10. **Else** generate an alert

end

It is based on two phases: learning and detection. For the learning phase, Polygraph scans the whole history data and constructs true sets and their corresponding true ranges. These two components are to be used for false alert detection. Line 3 and 4 explain how to apply Proposition 1 and 2 to detect false events. If an event E occurs at a host H and its parameter value is less than the minimum of H's true set, then E is considered to be a false event and Polygraph prevents it from generating an alert. Line 5 and 6 explain how to apply Proposition 3 to consider time dimension for false event detection. If E's parameter value is outside true range of H, then E is considered to be a false event. If E passes all these tests, then Polygraph considers it to be a true event, and allows generation of an alert.

5 Evaluation

Our empirical results are based on large and detailed datasets collected from globally distributed production environments serving real clients. There are four data components used in our experiments: alert policy, system performance metric, alert, and event datasets. With the exception of alert policies, all these are time-stamped data. For confidentiality reasons, we do not disclose information on clients, and detailed information of their datasets.

We collected 30-day datasets with around 60K events. We divide the datasets of system performance metrics, alerts, and events into six parts (5 days for each) based on their occurred time: older data are to be used for learning purpose, and recent data are for test purpose. We do not use cross-validation because policy threshold adjustment must be trained based on previously occurred events, not an arbitrarily chosen events. To show the effects of training data size on our alert threshold adjustment schemes, we use the first five datasets to make five differently sized training data (datasets of 5, 10, 15, 20, and 25 days) and the last part as test data.

In our experiments, we test the effect of Polygraph on events. Our original thought was to reduce the number of false alerts, but in reality it is hard to measure the effect of alert policy threshold adjustment as the reduction of the number of false alerts since an alert is usually generated by a mixture of events. Therefore, in our experiments, we test the effect of our framework on event datasets, which can indirectly measure the reduction in the number of false alerts by counting the reduction in the number of false events. Moreover, minimizing events reduce the communication overhead (from monitored servers to monitoring systems) in a highly distributed systems.

For the rest of this section, we explain characteristics of our four different type of datasets, and analyze experimental results of our threshold adaption schemes in various parameter settings. By default, we use 1 hour as a default true range threshold value.

5.1 Data Characteristics

Alert policies are pre-defined for each client. An alert policy dataset contains several fields including the server it is monitoring and predicate descriptions. Their parameter values have never been changed since they were originally deployed (typical in most environments).

Event data itself does not contain any information of whether it generated false alerts or true alerts. For this reason, we analyze alert data to get the class label of events. Alert datasets have their problem description and solution description as their record fields, and most of the solution descriptions contains useful information (pattern) that can be classified to either *true* or *false*. There are four different types of relations between events and alerts: (1) in general, an alert is generated by an event; (2) an alert might have more than one root events that

caused its generation (3) not every event contributes to generation of an alert; and (4) an event might be related to more than one alerts. As long as an event resulted in generating at least one alert with consistent solution (either true or false), we included it in our input data. A small portion of alerts missing their solutions were discarded in the experiments.

System performance metric data are collected at one minute intervals and are aggregated into different time windows such as 15 minute, 1 hour, 1 day, and further. In the experiments, we use the exact measure of the performance metrics in the alert policy, which is described in each event record.

We use two major alert policies, say P_1 and P_2, in our experiments to show the effectiveness of the Polygraph framework. Table 1 shows their characteristics. In fact, P_1 is the most frequently occurred alert policy among 360 alert policies that were triggered in our dataset, and most of its events are biased to generate false alerts. P_2 is also one of the most frequently triggered alert policy, but it almost uniformly generates true alerts and false alerts. P_1 will be a good example of how Polygraph can automatically and effectively tune the alert policy threshold, while the performance of P_2 indicates the reason Polygraph needs expert reviews to prevent abuse of automation.

Table 1: Characteristics of two different alert policies

Alert Policy	Count	Ratio(%)	True Events	False Events
P_1	23355	40.48	1026 (4.39%)	22329 (95.61%)
P_2	3344	5.80	1526 (45.63%)	1884 (56.34%)

5.2 Basic Alert Policy Threshold Adjustment

Both P_1 and P_2 alert policies did not have any gain when applying alert policy threshold adjustment schemes on the whole dataset, not considering each server or host separately. Based on further experiments which will be shown below, it does not mean that the current threshold is the optimized one, but it means the servers with same alert policy are not similar in the dataset.

5.3 Host-Based Alert Policy Threshold Adjustment

In Figure 4, we compare false alert detection rates of P_1 and P_2. For each alert policy, we show two lines: one is the false alert detection rate based on all servers, and the other is based only on the servers whose training data contains true events. The difference between those two lines indicates the false event detection rate

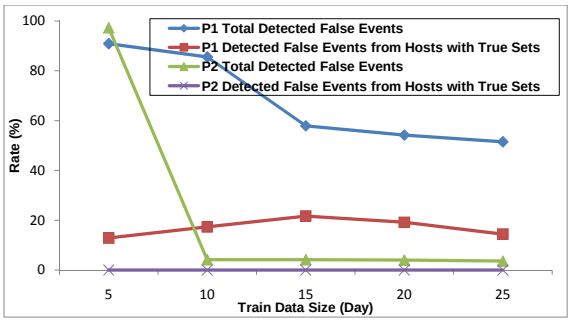

Figure 4: Host-based false event detection on P_1 and P_2

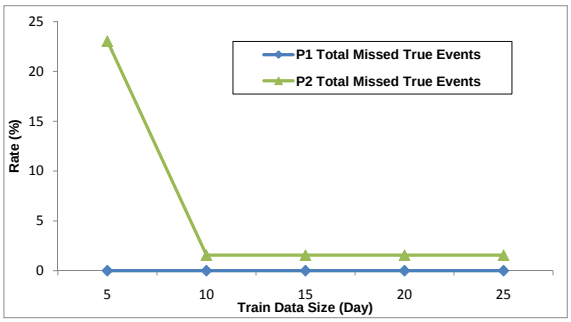

Figure 5: Missing true events for host-based false event detection on P_1 and P_2

of the servers whose training data do not have any true events. This type of servers exist mainly because of the following two reasons: either (1) training data is not big enough to hold all possible patterns; or (2) those servers do not need the given alert policy any more because it always generates false alerts. We can check it by looking at the true event missing rate described in Figure 5. (In this figure, we only showed two legends of P_1 and P_2 based on the whole training data. The missing rates of P_1 and P_2 based on the servers whose training data contains true events were all 0 for all training datasets.) In case of P_1, it does not miss any true events. Thus, some servers with P_1 alert policy would have no true set in their training data because of the latter reason. On the other hand, some servers with P_2 alert policy whose training data does not contain any true events also miss true events. Therefore, we can conclude that they require bigger training data to learn their true sets.

As easily seen in Figure 4 and 5, P_1 showed high rate of false event detection and did not miss any true events, while P_2 achieved low rate of false event detection with sometimes missing true events. For P_2, we see that at least 10 days of training data is needed for reliable performance. The spikes of P_2 in both figures are because of the increase of the number of servers whose training data does not contain true events. The purple line in Fig-

ure 5 declines at 5-day training dataset since quite a few servers lost all of their true events because of the dataset shrinkage. In general, we see that an alert policy detects more false events and misses more true events in total as the size of training data becomes smaller.

5.4 Host and Time-Based Alert Policy Threshold Adjustment

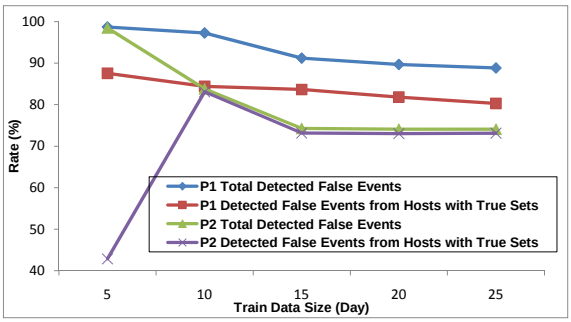

Figure 6: Host and time-based false event detection on P_1 and P_2

Figure 7: Missing true events for host and time-based false event detection on P_1 and P_2

Figure 6 shows false event detection rates of P_1 and P_2 when using both host- and time-based schemes. In Figure 7, we show true event missing rate of P_1 and P_2. As before, for each alert policy, we show two lines: one is the true alert missing rate based on entire training data, and the other is based only on the servers whose training data contains true events. For P_1, both lines were identical, so we only showed the total version of missing rates.

Compared with the results of the host-based scheme described in Figure 4, host-and-time-based scheme also shows similar trends except that its lines show higher false alert detection rates than those of host-based scheme. That is, host and time-based scheme achieves higher recall than host-based scheme. But on the other hand, it suffers from higher rate of missing true tickets, or in other words lower precision. For host-based scheme (described in Figure 5), most of the true event missing rates were 0 or similarly small for both alert policies, but now P_2 misses more than 10% of the true events. Note that it shows a big spike at the 5-day training dataset as before.

Based on the experimental results described above, P_1 can be safely tuned by Polygraph with no human interaction, but P_2 needs to be shown to the system administrator before deployment. One way to assure higher precision of not missing true events is to utilize the data occurred around the same day of the year for the past few years as additional training data. For example, a server might show similar patterns for Christmas season, every year. For this server, it will be helpful to use last year's Christmas season data as additional training data. It is based on the observation that our false event detection schemes never miss any true event if given perfect future knowledge of the whole dataset we used.

5.5 True Range Threshold Effect

In Polygraph, we only have one parameter, true range threshold, to be applied to host and time-based false event detection scheme. In Figure 8 and 9, we used 10-day training dataset by default and tested the effect of various true range threshold values. In Figure 9, we skipped to show the true alert missing rates of P_1 based on the servers whose training data contains true events since it is identical to P_1's total false event detection rates.

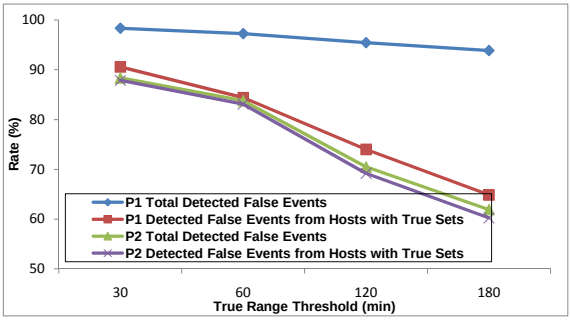

Figure 8: Host and time-based false event detection on P_1 and P_2 with various true range threshold values

As expected, in Figure 8, as the threshold value becomes bigger, the true ranges of P_1 and P_2 become larger which leads to the decrease of false event detection rate. Analyzing the slopes of both figures indicates that the setting of a true range threshold of P_2 has a big impact on false event detection and a small impact on missing true events, while for P_1 the performance of both false event detection and missing true tickets are mostly sta-

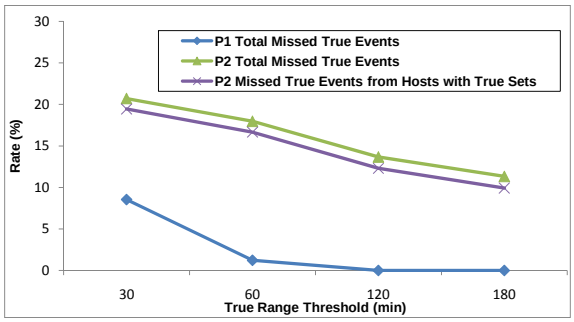

Figure 9: Missing true events for host and time-based false event detection on P_1 and P_2 with various true range threshold values

ble. One of the reason is because P_1 has quite a few servers that do not need it anymore. In that sense, P_1 is one of the desired alert policy that needs to be tuned with a big impact on the whole system performance.

6 Discussion

6.1 False Alert Detection and Policy Generation

In this paper, we proposed a framework for dynamically tuning monitoring policies in order to reduce the number of false alerts. The prototype proposed methods for automatic generation of threshold-based policies. In the following, we discuss several extensions that can be used to improve the effectiveness of the overall system.

Leverage operational data for monitoring policy tuning Polygraph framework enables the integration of a broader range of relevant operation data into alert management system. One of such operational data types is scheduled maintenance activities such as anti-virus scans and new user on-boarding which could generate significant workload on a server. Incorporating this type of information into monitoring rules/policies could help avoid false alerts that might be triggered during such activities. For instance, our analysis of the sample dataset used for a prototype evaluation shows that 20.3% of a customer's alerts were due to virus scan that caused higher CPU usage than the normal state. Adding active-application constraints can eliminate these type of false alerts. Another relevant operational data type is the SLA specifications and attainment information. It is yet to be explored how SLAs can be translated into individual monitoring policies appropriately to avoid over/under provisioning.

Emphasize more recent history When long history of data is available, such as in a typical production environment, false-alert detection is likely to exhibit poor quality if all data samples are given equal weight in the analysis because the detector would miss recognizing the most recent trends in false alert occurrences. In order to improve that decision quality, a weighted scheme can be employed to put more emphases (larger weights) on recent input. The method applies when our historical content is complete, not missing any false negatives (*i.e.*, not missing any true alerts).

6.2 Polygraph in Production Deployment

The integration of Polygraph with a monitoring infrastructure and service management tools like IBM Tivoli [9] will require the integration of novel approaches for ensuring scalability with the number of monitored components and deployed monitoring policies.

Policy deployment Polygraph analysis could result in the generation of new policies for a server profile that matches a very large number of servers. In order to avoid the disruption of the monitoring infrastructure, Polygraph must fully exploit the features and protocols of the scalable monitoring infrastructure. Also, Polygraph must inject the changes in a staged manner. The assessment on server-specific risks/costs due to delaying the policy deployment will be the base for deciding the staging order and timeline.

Change history considerations As discussed in 4.2, the infrastructure of a server (hardware, software, and workload, etc..) and the environment in which the server is running is not always static. System behavior may change over time. At the very beginning, when the servers and their sensors were initially set up, most of the servers were clustered together based on their workload types. As time passes, the initial policy is out-dated. Therefore, capturing all changes that server has experienced (for example, new patch/software installation, HD, CPU, Memory expansion, subnet change, etc..) and adjusting the policy becomes more important. As one of the enhancements in the near future, we will bring the server change history into consideration when recommending policy changes.

Leverage server similarity for monitoring policy tuning One of the insights from our experiments in Section 5 is the potential benefit of grouping similar servers in alert policy tuning. This is particularly helpful in the cases when the train dataset collected on an individual server does not have sufficient data points for some rare events. Grouping similar servers will provide a better train dataset, hence better policy tuning. A simple example is the clustered load-sharing servers for applications

with high traffic volumes, which have the same server configuration and the same workload characteristics. For more general cases, it will be challenging to define a server similarity measure that may be composed of not only a server resource profile but also the workload characteristics. For Polygraph system implementation, it requires integration with the data sources that maintain up-to-date server profile information and workload history.

7 Conclusion

With the increasing complexity and scale of modern data centers, efficient and cost-effective management of incidents and failures have become an important optimization target for IT service providers. Alerts can originate from anywhere in the end-to-end service management stack, e.g. from servers, storage, networks, middleware, and applications. Our data collected from a large IT service delivery environment indicates that often a large portion of these alerts can be false positives, which can safely be ignored, and with suitable techniques can be eliminated from the environment. Our system, called Polygraph, uses historical incidents and their resolutions to learn which alerts can be safely labeled as false; and correlates the alerts/incidents data with other information such as system vitals and server similarities to update policies that trigger these alerts in the first place. In our experiments with real-life traces from a large service delivery environment that includes many different types of servers and application architectures, Polygraph performs very well in reducing false alerts while keeping false negatives to a low level. Polygraph achieves this via both host-based and time-based tuning of the monitoring policies.

There are several areas of future work. First, we plan to extent Polygraph by incorporating operational context, by automatically identifying durations when certain processes are kickstarted on a server generating a temporary spike in resource consumption. Second, we plan to investigate tuning the monitoring policies to a less stricter or more relaxed threshold values to improve the balance between false alert reduction and potential SLO violation. Finally, by giving more weight to recent incident history, we anticipate that the quality of false alert detection in Polygraph will increase, especially when the historical content is complete, i.e. not missing any false negatives.

References

[1] C. MILLS ET AL. Method and apparatus for alert prioritization on high value end points. *US patent application US2009/0271792A1* (2009).

[2] CHANDOLA, V., BANERJEE, A., AND KUMAR, V. Anomaly Detection: A Survey. *ACM Computing Surveys* (2009).

[3] D. BREIGAND, E. H., AND SHEHORY, O. Automated and Adaptive Threshold Setting: Enabling Technology for Autonomy and Self-Management. *Proceedings of the Second International Conference on Automatic Computing, ICAC '05* (2005).

[4] DRUMMOND, C., AND HOLTE, R. Learning to Live with False Alarms. *Proceedings of the KDD Data Mining Methods for Anomaly Detection Workshop* (2005).

[5] F. XIAO AND S. JIN AND S. SHI AND X. LI. A Novel Data Mining-Based Method for Alert Reduction and Analysis. *Journal of Networks, Vol 5, No 1* (2010).

[6] GOETHALS, B., AND ZAKI, M. J. Advances in frequent itemset mining implementations: report on FIMI'03. *SIGKDD Explor. Newsl.* (2009).

[7] HEWLETT-PACKARD COMPANY. HP ServiceCenter software. *HP website* (2011).

[8] IBM CORPORATION. Tivoli Monitoring. *http://www-01.ibm.com/software/tivoli/products/monitor/* (2010).

[9] IBM CORPORATION. Tivoli Software. *http://www-01.ibm.com/software/tivoli/* (2010).

[10] J.S. SHIRABAD, T. L., AND MATWIN, S. Supporting software maintenance by mining software update records. *IEEE International Conference on Software Maintenance* (2001).

[11] JULISCH, K. Mining alarm clusters to improve alarm handling efficiency. *ACSAC '01 Proceedings of the 17th Annual Computer Security Applications Conference* (2001).

[12] JULISCH, K. Clustering Intrusion Detection Alarms to Support Root Cause Analysis. *ACM Transactions on Information and System Security TISSEC* (Nov 2003).

[13] MOCKUS, A., AND WEISS, D. Predicting Risk of Software Changes. *Bell Labs Technical Journal* (2002).

[14] N. JAN, S. LIN, S. T., AND LIN, N. A decision support system for constructing an alert classification model . *Expert Systems with Applications Vol. 36, No. 8* (Oct. 2009).

[15] NAGIOS ENTERPRISES. Enterprise management solutions. *http://www.nagios.org/* (2010).

[16] S. MENG, T. W., AND LIU, L. Monitoring continuous state violation in datacenters: Exploring the time dimension. *IEEE 26th International Conference on Data Engineering (ICDE)* (2010).

[17] SOLEIMANI, M., AND GHORBANI, A. Critical Episode Mining in Intrusion Detection Alerts. *Proceedings of Communication Networks and Services Research Conference (CNSR)* (2008).